UNLIMITED

RSRE
MEMORANDUM No. 4436

ROYAL SIGNALS & RADAR ESTABLISHMENT

FORMAL SEMANTIC DEFINITION OF ELLA TIMING

Authors: M G Hill, E V Whiting & J D Morison

PROCUREMENT EXECUTIVE,
MINISTRY OF DEFENCE,
RSRE MALVERN,
WORCS.

RSRE MEMORANDUM No. 4436

UNLIMITED

CONDITIONS OF RELEASE

0088204 BR-115854

********** DRIC U

********** DRIC Y

Reports quoted are not necessarily available to members of the public or to commercial organisations.

Contents

THIS PAGE IS LEFT BLANK INTENTIONALLY

1 Introduction

In this document we describe work that has resulted from the collaborative projects ESPRIT 'SPRITE' [1-3] and IED 'Formal Verification' for the formal definition of the ELLA^{TM} timing model. We begin by describing informally the semantics of ELLA delays, then the formal definition of ELLA timing is presented using predicate calculus. We also define the predicate for the new ELLA 'sample and hold' timing primitive used in the retiming of ELLA circuits. The concept of hierarchically timescaled regions is presented. Examples of the use of the new timing primitive in hierarchical timescaling are given.

2 ELLA Time

The ELLA time model is based on the use of delay behavioural primitives. These primitives are used to define leaf nodes whose sole property is to delay a value passing through that node. The ELLA simulator has an internal clock that starts from zero and increases in increments of one, and the delay time unit is defined in terms of this clock. The relationship between simulation time and real time is left for the user to interpret. The delay primitives serve a number of purposes, for example describing propagation delays, providing memory, or producing high-level timing functions similar to those in register transfer languages. There are two delay primitives, DELAY and IDELAY, each of which must be the sole contents of a function body. Examples of these primitives are

```
FN DELAY_1 = (type) -> type: DELAY(?type, n).

FN DELAY_2 = (type) -> type: DELAY(?type, 2, ?type, 5).

FN DELAY_3 = (type) -> type: IDELAY(?type, 2).
```

where DELAY_1 is a transport delay and delays the input value by 'n' time units, where 'n' is an integer. Any signal that remains stable for less than the delay time, i.e 'n', is replaced by the basic value '?type'. Note that if n=1 then DELAY_1 is a pure unit delay. DELAY_2 is an ambiguity delay where at zero time it has the value ?type. After two time units the value will change to its second type value (here it will stay at '?type'), and then after a further three time units, i.e. five units since time zero, the input to the function appears at the output. With this type of delay if the input is stable for less than five time units the output becomes '?type' two time units after the first change, and it remains '?type' until five time units after the second change. DELAY_3 is an inertial delay which delays the input by two time units and outputs '?type' for the first two time units. With an inertial delay any signal that remains stable for less than the delay time (two units in this case) is filtered out.

Consider the following example where

```
TYPE type = NEW (h | l).
```

with the input signal to the three delays defined above given as

```
                    h      hhhhhhhh
INPUT   :   lll llllll
time    :   01234..........
```

then their corresponding outputs are

```
                              hhhhhh
DELAY_1:      lll llllll
{n=2}       ??    ?

                                 hhhh
DELAY_2:              ll
            ????????  ???

                              hhhhhh
DELAY_3:      lllllllll
            ??
```

where DELAY_1 has been instantiated for the value n=2.

The above description provides a very informal description of the behaviour of delays. In the next section we define the delay behaviour rigorously by means of predicate calculus.

3 ELLA Delay Predicates

A brief overview of the predicate calculus terminology used in this section is given in appendix A. For a more in-depth description the reader is referred to [9] and [10].

3.1 Ambiguity Delay

Consider the following ambiguity delay function

FN DEL = (type: in) → type: DELAY(val1, m, val2, n).

where $0 < m \leq n$. The semantics for this delay function are given informally as

- The initial value of the function is 'val1'. If there exists a sequence of n equal input values occurring between the current time and m+n-1 units of time ago then the output of the function is the input value n units of time ago, otherwise the output is 'val2'.

The semantics can be formally expressed as

$$
\begin{array}{lll}
(\forall t)\ \text{IF} & t < 0 & \\
\quad \text{THEN} & in(t) = val1 & \\
\quad \text{ELSE} & \text{IF } \exists j\ \forall i\ in(t-n+i-j) = in(t-n) & (1) \\
\quad & \quad \text{THEN } DEL\ in(t) = in(t-n) & \\
\quad & \quad \text{ELSE } DEL\ in(t) = val2 &
\end{array}
$$

where $1 \leq j \leq m$, $1 \leq i \leq n$. It is necessary to initialise the input signal for times less than zero since the remainder of the predicate requires values of 'in' from a time of $t = -n - m + 1$. Without this initialisation of condition on the input signal a complex set of conditions would be needed to capture the behaviour of DEL.

It can be noted that by using the following notation

$$IF\ p\ THEN\ q\ ELSE\ r \ \equiv\ [\ p \Longrightarrow q \mid r\ ] \tag{2}$$

the above expression may be rewritten as

$$(\forall t)\ [\ t < 0 \Longrightarrow in(t) = val1 \mid \tag{3}$$

$$[\ \exists j\ \forall i\ in(t - n + i - j) = in(t - n) \Longrightarrow DEL\ in(t) = in(t - n) \mid DEL\ in(t) = val2\ ]\ ]$$

These notations are interchangeable and in this document use will be made of both formats.

The outer ELSE clause of (1) determines the output value for the delay function. It does this by examining a sequence of n equal input values starting with the range [t-1 .. t-n] and finishing with the range [(t-n-m+1) .. t-m]. Pictorially this can be seen in the following diagram which illustrates alternatives for the input signal to the delay which have a pulse length of at least size n

```
{t-}   n+m-1          n+1 n n-1          m          2 1     j=
         ?              ?  x  x  x x x x  x  x x x x  x x     1
         ?           ?  x  x  x  x x x x  x  x x x x  x 0     2
         ?        ? x  x  x  x  x x x x  x  x x x x  0 ?     3
         :        : :  :  :  :  : : : :  :  : : : :  : :     :
       ? x    x x x x  x  x  x  x x x x  x  0 ? ? ?  ? ?     m
```

where "x" indicates equal values, "0" a value different from "x", and "?" indicates an "x", "0" or another value. It can be noted that although the value in(t-n) was chosen in the second 'THEN' arm it could equally have been the value in(t-k) where $m \leq k \leq n$ since they are all the same value.

Up to this point we have assumed that in(t) has a known value so that a sequence of n equal values can be found. However in ELLA a signal, such as the input to the delay function, can have a value of '?type', which represents the ELLA unknown value. In this case the above predicate has to be extended to allow for the following case

- If a sequence of n values are found which are a combination of a single value and the ELLA unknown value then the delay function should return the unknown value.

This can be formally expressed as

$$IF\ \exists j\ \forall i\ in(t-n+i-j) = (in(t-n) \vee ?type)\ THEN\ DEL\ in(t) = ?type \qquad (4)$$

Combining this with (1) gives

$$\begin{array}{lll}
(\forall t)\ \text{IF} & t < 0 & \\
\quad \text{THEN} & in(t) = val1 & \\
\quad \text{ELSE} & \text{IF} & \exists j\ \forall i\ in(t-n+i-j) = in(t-n) \\
& \quad \text{THEN} & DEL\ in(t) = in(t-n) \\
& \quad \text{ELSE} & \text{IF}\ \exists j\ \forall i\ in(t-n+i-j) = (in(t-n) \vee ?type) \\
& & \quad \text{THEN}\ DEL\ in(t) = ?type \\
& & \quad \text{ELSE}\ DEL\ in(t) = val2
\end{array} \qquad (5)$$

where $1 \leq j \leq m,\ 1 \leq i \leq n$

For the particular case of m = n = 1, i.e. Pure Unit Delay, we see that in (5) i = j =1, thus $in(t-n+i-j) = in(t-n)$ and hence the predicate reduces to

$$(\forall t)\ [\ t < 0 \Longrightarrow in(t) = val1 \mid DEL\ in(t) = in(t-1)\ ] \qquad (6)$$

For the particular case of m = 0 we need to redefine the range of 'j' to be $MIN(1,m) \leq j \leq m$ so that $m = 0 \Longrightarrow j = 0$, thus giving

$$\begin{array}{lll}
(\forall t)\ \text{IF} & t < 0 & \\
\quad \text{THEN} & in(t) = val1 & \\
\quad \text{ELSE} & \text{IF} & \forall i\ in(t-n+i) = in(t-n) \\
& \quad \text{THEN} & DEL\ in(t) = in(t-n) \\
& \quad \text{ELSE} & \text{IF}\ \forall i\ in(t-n+i) = (in(t-n) \vee ?type) \\
& & \quad \text{THEN}\ DEL\ in(t) = ?type \\
& & \quad \text{ELSE}\ DEL\ in(t) = val2
\end{array} \qquad (7)$$

where $1 \leq i \leq n$.

In summary, the predicate for the ambiguity delay function for $0 \leq m \leq n$ is given by

$$\begin{array}{lll}
(\forall t)\ \text{IF} & t < 0 & \\
\quad \text{THEN} & in(t) = val1 & \\
\quad \text{ELSE} & \text{IF} & \exists j\ \forall i\ in(t-n+i-j) = in(t-n) \\
& \quad \text{THEN} & DEL\ in(t) = in(t-n) \\
& \quad \text{ELSE} & \text{IF}\ \exists j\ \forall i\ in(t-n+i-j) = (in(t-n) \vee ?type) \\
& & \quad \text{THEN}\ DEL\ in(t) = ?type \\
& & \quad \text{ELSE}\ DEL\ in(t) = val2
\end{array} \qquad (8)$$

where $MIN(1,m) \leq j \leq m,\ 1 \leq i \leq n$.

3.2 Transport Delay

The transport delay is a particular case of the ambiguity delay where m = n and val1 = val2. Thus consider the transport delay function

FN DEL_T = (type: in) → type: DELAY(val, m).

Then from (8) we have that the predicate for this delay function is

$$
\begin{array}{l}
(\forall t)\ \text{IF}\ \ t < 0 \\
\qquad \text{THEN}\ \ in(t) = val \\
\qquad \text{ELSE}\ \ \text{IF}\ \ \exists j\ \forall i\ in(t - m + i - j) = in(t - m) \\
\qquad\qquad \text{THEN}\ DEL\_T\ in(t) = in(t - m) \\
\qquad\qquad \text{ELSE}\ \ \text{IF}\ \ \exists j\ \forall i\ in(t - m + i - j) = (in(t - m) \vee ?type) \\
\qquad\qquad\qquad \text{THEN}\ DEL\_T\ in(t) = ?type \\
\qquad\qquad\qquad \text{ELSE}\ DEL\_T\ in(t) = val
\end{array}
\tag{9}
$$

where $0 < m$, $1 \leq i, j \leq m$.

3.3 General Delay

Consider the following delay function

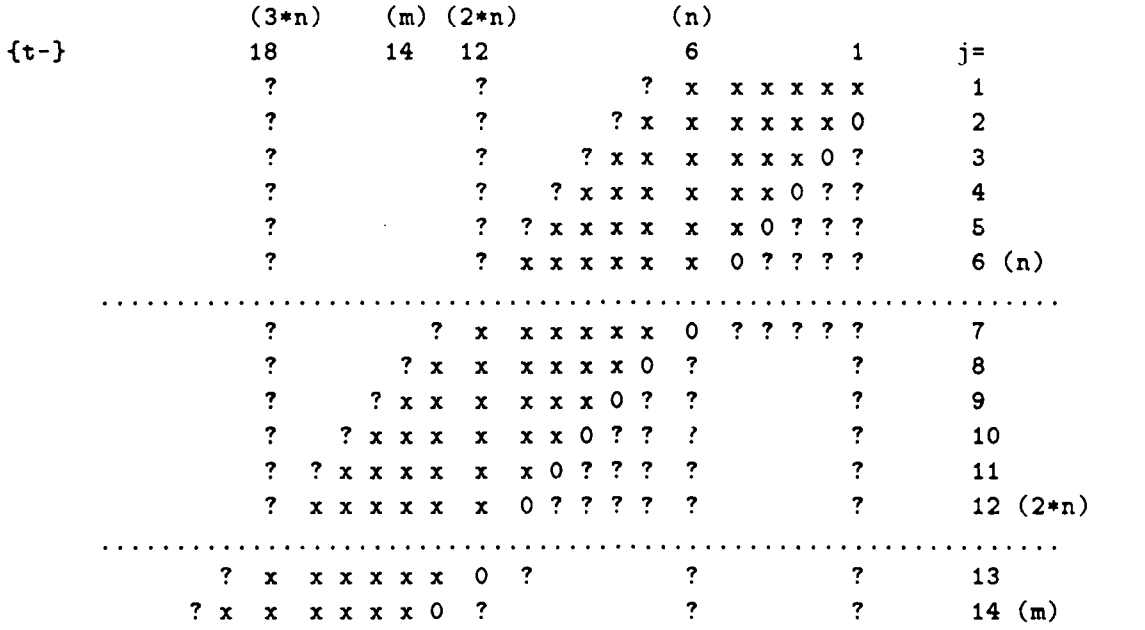
FN DEL_G = (type: in) → type: DELAY(val1, m, val2, n).

where $m > n$. The semantics for this delay function are the same as for the ambiguity delay i.e.

- The initial value of the function is 'val1'. If there exists a sequence of n equal input values occurring between the current time and m+n-1 units of time ago then the output of the function is the input value n units of time ago, otherwise the output is 'val2'.

It should be noted that at present ELLA does not support the full range of general delay functions.

To gain insight into the general delay case where $m > n$ consider the following example where n = 6 and m = 14. The following alternatives for the input signal are possible (for pulses of at least size 'n')



where "x" indicates equal values, "0" a value different from the "x" value, and "?" indicates an "x", "0" or another value.

The division of the diagram shows that for values of j between 1 and n the output value will always be available at t-n. Similarly between j=n+1 and j=2*n the output value will be available at t-(2*n). By using the observation that intervals of 'n' units can be considered in turn we can extend the predicate of the ambiguity delay by means of the introduction of a variable λ which controls the particular block of n signals that are being considered. Thus leaving '?type' aside for the moment we have that a predicate definition for the general delay is

$$
\begin{array}{l}
(\forall t)\ \text{IF}\ \ t < 0 \\
\qquad \text{THEN}\ \ in(t) = val1 \\
\qquad \text{ELSE}\ \ (\exists \lambda)\ \text{IF}\ F(\lambda, t)\ \wedge\ \forall \mu\ \neg F(\mu, t) \\
\qquad\qquad\qquad \text{THEN}\ DEL\_G\ in(t) = in(t - \lambda.n) \\
\qquad\qquad\qquad \text{ELSE}\ \ DEL\_G\ in(t) = val2
\end{array}
\tag{10}
$$

where

$$
\begin{array}{l}
F(\lambda, t)\ \equiv\ \exists j\ \forall i\ in(t - \lambda.n + i - j) = in(t - \lambda.n) \\
\qquad\qquad 1 \leq \lambda \leq INT(m, n) \\
\qquad\qquad 1 \leq i, j \leq n \\
\qquad\qquad 1 \leq \mu < \lambda
\end{array}
\tag{11}
$$

and 'INT(m, n)' is the integer greater than or equal to m/n.

Examining (10) we see that $F(\lambda, t)$ attempts to find a sequence of n equal values and $\neg F(\mu, t)$ ensures that the value of λ is minimum.

Extending this predicate to allow for '?type' gives the following result for the general delay predicate

$$
\begin{array}{l}
(\forall t)\ \text{IF}\ \ t < 0 \\
\qquad \text{THEN}\ \ in(t) = val1 \\
\qquad \text{ELSE}\ \ (\exists \lambda)\ \text{IF}\ G(\lambda, t)\ \wedge\ \forall \mu\ \neg G(\mu, t) \\
\qquad\qquad\qquad \text{THEN IF}\ \ F(\lambda, t) \\
\qquad\qquad\qquad\qquad\qquad \text{THEN}\ DEL\_G\ in(t) = in(t - \lambda.n) \\
\qquad\qquad\qquad\qquad\qquad \text{ELSE}\ DEL\_G\ in(t) = ?type \\
\qquad\qquad\qquad \text{ELSE}\ \ DEL\_G\ in(t) = val2
\end{array}
\tag{12}
$$

where

$$
\begin{array}{c}
F(\lambda, t)\ \equiv\ \exists j\ \forall i\ in(t - \lambda.n + i - j) = in(t - \lambda.n) \\
G(\lambda, t)\ \equiv\ \exists j\ \forall i\ in(t - \lambda.n + i - j) = (in(t - \lambda.n) \vee ?type) \\
1 \leq \lambda \leq INT(m, n) \\
1 \leq i, j \leq n \\
1 \leq \mu < \lambda
\end{array}
\tag{13}
$$

and 'INT(m, n)' is the integer greater than or equal to m/n.

It can be noted that this predicate can also be used to describe the case where $0 \leq m \leq n$ by simply taking j to have the range $MIN(1, m) \leq j \leq MIN(m, n)$ and extending the definition of INT so that INT(0,n)=1. From (13) $\lambda = 1$ when $m < n$ and in that case there is no μ such that $1 \leq \mu < \lambda$. Thus (12) reduces to

$$
\begin{array}{l}
(\forall t)\ \text{IF}\ \ t < 0 \\
\qquad \text{THEN}\ \ in(t) = val1 \\
\qquad \text{ELSE}\ \ \text{IF}\ \exists j\ \forall i\ in(t - n + i - j) = (in(t - n) \vee ?type) \\
\qquad\qquad\qquad \text{THEN IF}\ \ \exists j\ \forall i\ in(t - n + i - j) = in(t - n) \\
\qquad\qquad\qquad\qquad\qquad \text{THEN}\ DEL\_G\ in(t) = in(t - n) \\
\qquad\qquad\qquad\qquad\qquad \text{ELSE}\ DEL\_G\ in(t) = ?type \\
\qquad\qquad\qquad \text{ELSE}\ \ DEL\_G\ in(t) = val2
\end{array}
\tag{14}
$$

where $MIN(1, m) \leq j \leq m$, $1 \leq i \leq n$ and (14) is functionally equivalent to (8).

In summary, the predicate for a delay function with $0 \leq m$, $1 \leq n$ can be given as

$$
\begin{array}{l}
(\forall t)\ \text{IF}\ \ t \prec 0 \\
\qquad \text{THEN}\ \ in(t) = val1 \\
\qquad \text{ELSE}\ \ (\exists \lambda)\ \text{IF}\ G(\lambda, t)\ \wedge\ \forall \mu\ \neg G(\mu, t) \\
\qquad\qquad\qquad \text{THEN IF}\ \ F(\lambda, t) \\
\qquad\qquad\qquad\qquad\qquad \text{THEN}\ DEL\_G\ in(t) = in(t - \lambda.n) \\
\qquad\qquad\qquad\qquad\qquad \text{ELSE}\ DEL\_G\ in(t) = ?type \\
\qquad\qquad\qquad \text{ELSE}\ \ DEL\_G\ in(t) = val2
\end{array}
\tag{15}
$$

where

$$
\begin{array}{c}
F(\lambda, t)\ \equiv\ \exists j\ \forall i\ in(t - \lambda.n + i - j) = in(t - \lambda.n) \\
G(\lambda, t)\ \equiv\ \exists j\ \forall i\ in(t - \lambda.n + i - j) = (in(t - \lambda.n) \vee ?type) \\
1 \leq \lambda \leq INT(m, n) \\
1 \leq i \leq n \\
MIN(1, m) \leq j \leq MIN(m, n) \\
1 \prec \mu < \lambda
\end{array}
\tag{16}
$$

and 'INT(m, n)' is the integer greater than or equal to MAX(1, m/n).

3.4 Interial Delay

Consider the following interial delay function

FN IDEL = (type: in) → type: IDELAY(val, n).

The semantics of this delay function are given informally as

- The initial value of the function is 'val'. The function delays its input signal by 'n' units, and if the signal changes within that time then the changes are filtered out.

One formal interpretation of the semantics of this function is given by

$$
\begin{array}{l}
(\forall t)\ \text{IF}\ \ t < 0 \\
\qquad \text{THEN}\ \ in(t) = val \\
\qquad \text{ELSE}\ \ \text{IF}\ \ \forall i\ in(t - 1) = in(t - 1 - i) \\
\qquad\qquad\qquad\ \ \text{THEN}\ IDEL\ in(t) = in(t - n) \\
\qquad\qquad\qquad\ \ \text{ELSE}\ \ IDEL\ in(t) = IDEL\ in(t - 1)
\end{array}
\tag{17}
$$

where $1 \leq i \leq n$ and this recursive call shows the memory of IDEL.

A formal definition of this function can also be obtained by using the predicate for the general delay. In this case 'm' in (15) is allowed to tend towards infinity. Since this means that all blocks of 'n' signals are considered, even those which occurred before simulation began, the condition $G(\lambda, t)$ will always be met eventually (all input signals for $t < 0$ have the same value). Hence the predicate (15) becomes, on letting $m \rightarrow \infty$

$$
\begin{array}{lll}
(\forall t)\ \text{IF} & t < 0 & \\
\quad \text{THEN} & in(t) = val & \\
\quad \text{ELSE} & (\exists \lambda)\ \text{IF}\ G(\lambda, t) \wedge \forall \mu\ \neg G(\mu, t) & \\
& \quad \text{THEN IF}\ F(\lambda, t) & \\
& \quad\quad \text{THEN}\ IDEL\ in(t) = in(t - \lambda.n) & \\
& \quad\quad \text{ELSE}\ IDEL\ in(t) = ?type &
\end{array}
\tag{18}
$$

where

$$
\begin{aligned}
F(\lambda, t) &\equiv \exists j\ \forall i\ in(t - \lambda.n + i - j) = in(t - \lambda.n) \\
G(\lambda, t) &\equiv \exists j\ \forall i\ in(t - \lambda.n + i - j) = (in(t - \lambda.n) \vee ?type) \\
& 1 \leq \lambda < \infty \\
& 1 \leq i, j \leq n \\
& 1 \leq \mu < \lambda
\end{aligned}
\tag{19}
$$

4 Timescaling

The timescaling enhancements for ELLA come in two parts. First the introduction of a new language primitive, second, the introduction of hierarchic time regions. The language primitive is a sample-and-hold construct which will allow synchronous descriptions to use ELLA time to describe clock periods other than one per time tick. The hierarchic approach will allow users to wrap up ELLA functions into regions which operate with clock periods either faster or slower than the surrounding region. These two features are related by a transformation and the simulator makes use of this fact.

For example, a hierarchically faster region running at four times the rate of the outer region, say, would be transformed into a region which has its inputs held constant for four time units and its output sampled ever four time units. The outer region would then have its delay times multiplied by four. Thus both regions would appear to operate within the same time frame, however only the inner region would change each time unit (the outer region effectively changing only every four time units). Thus users have at their disposal a new timing primitive which can be accessed either explicitly or implicitly. For a complete description of the enhancements and the implications on the ELLA system the reader is referred to [4-7].

4.1 Sample Primitive

The new sample-and-hold primitive occurs in the same syntactic position as the DELAY primitive, that is it is the sole contents of a function body. The syntax of the new primitive is of the form [7]

SAMPLE(interval_size, initial_value, skew)

or

SAMPLE(interval_size)

where if ommitted 'initial_value' and 'skew' default to ?type and zero, respectively. The 'skew' value determines the sample point and its value must be less than the interval_size.

Consider the following macro function which provides a mechanism for instantiating the SAMPLE primitive.

MAC SAMPLE_HOLD {INT interval, CONST(TYPE type) init, INT skew, }
= (type: in) → type: SAMPLE(interval, init, skew).

The semantics of this function are given informally as

- At a time 't' if (t-skew) is some multiple of the interval size then take the current input value, otherwise the output remains unchanged.

The semantics are defined formally by the following predicate.

$$(\forall t)\ [\ t < 0 \Longrightarrow in(t) = init \mid SAMPLE\_HOLD\ in(t) = in(t-((t-skew)MODinterval))\ ] \tag{20}$$

Thus consider the following example

```
TYPE ty = NEW (t1|t2|t3|t4|t5|t6|t7|t8|un).

FN SMP = (ty) -> ty: SAMPLE(4,un, 2).
```

In this case SMP samples the input signal every 4 units with the sample point occurring after 2 units. A typical simulation run then gives the following result

```
TIME   :  0   1   2   3   4   5   6   7   8   9  10  11

input  :  t1  t2  t3  t4  t5  t6  t7  t8  t1  t2  t3  t4
output :  un  un  t3  t3  t3  t3  t7  t7  t7  t7  t3  t3
```

4.2 Hierarchical Timing

Hierarchical timing is obtained by the use of the FASTER and SLOWER constructs. Both constructs appear in the same syntactic position as the SAMPLE primitive, i.e. the sole contents of a function body, and they instantiate a function to run at a simulation rate faster or slower (respectively) than the enclosing region.

The syntax for the FASTER construct is

FASTER(function_name, interval_size, initial_value, skew)

or

FASTER(function_name, interval_size)

where, 'function_name' is the name of the function which is to have its clock rate set at 'interval_size' times faster than the surronding region. The parameter 'initial_value' determines the initial output value of the faster region and 'skew' sets the input sample position, if omitted 'initial_value' and 'skew' default to ?type and zero respectively. A similar syntax follows for SLOWER i.e.

SLOWER(function_name, interval_size, initial_value, skew)

or

SLOWER(function_name, interval_size)

again if omitted 'initial_value' and 'skew' default to ?type and zero respectively. The restriction of the size of 'skew' for FASTER and SLOWER is the same as for SAMPLE.

Now consider the following macro functions

```
MAC FASTER_F
    {FN (TYPE type) → type: F, INT interval, CONST(type) init, INT skew, }
                        = (type: in) → type: FASTER(F, interval, init, skew).
```

and

```
MAC SLOWER_S
    {FN (TYPE type) → type: S, INT interval, CONST(type) init, INT skew, }
                        = (type: in) → type: SLOWER(S, interval, init, skew).
```

The predicates for these functions are defined in terms of the transformations which replace the FASTER/SLOWER constructs with an appropriate SAMPLE function. Thus the predicates for FASTER_F is given by

$$(\forall t)\ [\ t < 0 \Longrightarrow in(t) = init \mid FASTER\_F\ in(t) = SAMPLE\_HOLD(F\ in(t))\ ] \quad (21)$$

and the predicate for SLOWER_S is given by

$$(\forall t)\ [\ t < 0 \Longrightarrow in(t) = init \mid SLOWER\_S\ in(t) = S(SAMPLE\_HOLD\ in(t))\ ] \quad (22)$$

In both cases 'time' is defined to be the common time base which is the least common multiple of the inner faster/slower region and the enclosing region.

4.3 An Example Transformation

In this section a very simple circuit is shown to demonstrate how the faster and slower features are transformed to circuits with sample primitives. Consider the following ELLA circuit which is shown in figure 2.

```
TYPE a = NEW (a1|a2).

FN DEL = (a) -> a:DELAY(?a,2).

FN F = (a:in) -> a: DEL in.

FN S = (a:in) -> a: DEL in.

FN FAST = (a) -> a: FASTER(F,4,a1,2).

FN SLOW = (a) -> a: SLOWER(S,3,a2,1).

FN MAIN = (a:in) -> a: DEL(SLOW(DEL(FAST in))).
```

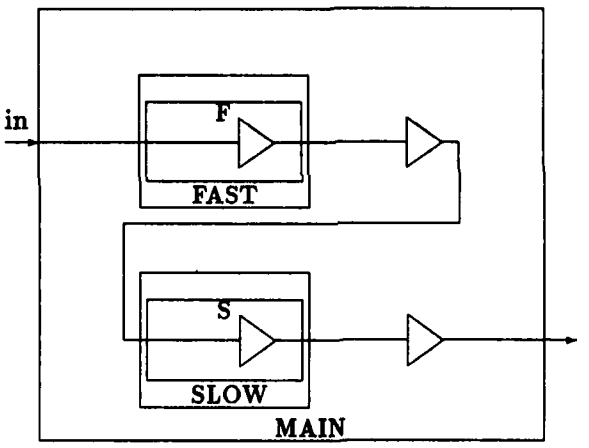


Figure 2: Faster and Slower

This circuit has little practical use and is given merely to illustrate the transformation procedure. The functions FAST and SLOW are transformed to remove the FASTER and SLOWER constructs using one of ELLAs in-built transformations. The resulting circuit can then be described in the following way.

```
MAC NEW_DEL{INT period} = (a) -> a: DELAY(?a, period*2).

FN F_SAMPLE = (a) -> a: SAMPLE(4, a1, 2).

FN S_SAMPLE = (a) -> a: SAMPLE(12, a2, 4).

FN NEW_FAST = (a:in) -> a: F_SAMPLE(NEW_DEL{1}in).
```

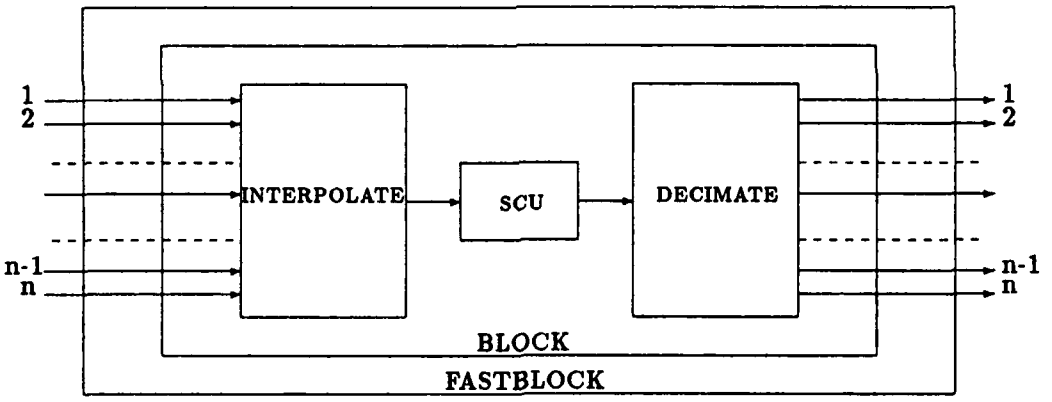
```
FN NEW_SLOW = (a:in) -> a: NEW_DEL{12}(S_SAMPLE in).

FN NEW_MAIN = (a:in) -> a:
              NEW_DEL{4}(NEW_SLOW(NEW_DEL{4}(NEW_FAST in))).
```

The functions FAST and SLOW have now been transformed so that they have a SAMPLE function at the end and beginning of their respective new forms. Because SAMPLE is a core primitive, functions F_SAMPLE and S_SAMPLE are needed to instantiate particular instances. It can be seen that the delays external to the timescaled regions have also been transformed. If function NEW_MAIN is now simulated its 'clock' will be that of the common timebase.

4.4 Retiming Example

In this section we apply the new retiming constructs to a simple 'parallel to serial to parallel' circuit.

Consider the following circuit diagram.



which corresponds to the following ELLA description

```
TYPE word = ...

MAC INTERPOLATE = ([INT n]word:input) -> word:
( SEQ
     TYPE intcount = NEW ic/(1..n);
     FN INC = (intcount:in) -> intcount:
                 ARITH IF in = n THEN 1 ELSE in+1 FI;
     PVAR count ::= ic/1;
     LET out = input[[count]];
     count :=  INC count;
     OUTPUT out
).
```

```
MAC DECIMATE {INT n} = (word:newvalue) -> [n]word:
( SEQ
     TYPE intcount = NEW ic/(1..n);
     FN INC = (intcount:in) -> intcount:
                 ARITH IF in = n THEN 1 ELSE in+1 FI;
     PVAR count ::= ic/1;
     PVAR out ::= [n]zeroword;
     LET pastout = out;
     out[[count]] := newvalue;
     count :=  INC count;
     OUTPUT pastout
).

FN SCU = (word:input) -> word: ...

FN BLOCK = ([n]word:input) -> [n]word:
( LET inter = INTERPOLATE input.
  OUTPUT DECIMATE {n} (SCU inter)
).

FN FASTBLOCK = ([n]word: in) -> [n]word: FASTER(BLOCK, n).
```

where the body of function 'SCU' has been left unspecified, this function represents some form of serial computation unit. The keyword 'FASTER' in the function FASTBLOCK signifies entry into a region where the internal clock of the function 'BLOCK' is changing 'n' times faster than the outer clock. The circuit is thus modelling a parallel to serial (INTERPOLATE), a serial computation unit (SCU), and a serial to parallel (DECIMATE) function, where 'n' parallel signals are transformed into 'n' sequential signals which then pass through a computation unit running at 'n' times the outer rate, before being re-grouped in parallel. This circuit follows the format of the Silage [8] interpolate and decimate constructs.

The effect of the faster region on the way signals are handled is shown in the following example where SCU is taken as

```
FN SCU = (word:input) -> word:input.
```

Consider the case where n=4, and 'word' is an enumeration type defined by

```
TYPE word = NEW (a1 | a2 | a3 | a4 | b1 | b2 | b3 | b4 | bn ).
```

Then simulating the function 'BLOCK' with the input

```
input = (a1, a2, a3, a4)      for t = 0..3

input = (b1, b2, b3, b4)      for t = 4..7

input = (bn, bn, bn, bn)      for t = 8
```

gives the following result

```
TIME      BLOCK          inter (internal node of
  0     ?  ?  ?  ?          a1        function BLOCK)
  1     a1 ?  ?  ?          a2
  2     a1 a2 ?  ?          a3
  3     a1 a2 a3 ?          a4
  4     a1 a2 a3 a4         b1
  5     b1 a2 a3 a4         b2
  6     b1 b2 a3 a4         b3
  7     b1 b2 b3 a4         b4
  8     b1 b2 b3 b4         bn
```

Simulating the function FASTBLOCK with the input

```
input = (a1, a2, a3, a4)      for t = 0  (i.e t_inner = 0..3)

input = (b1, b2, b3, b4)      for t = 1  (i.e t_inner = 4..7)

input = (bn, bn, bn, bn)      for t = 2
```

gives the following result

```
TIME        FASTBLOCK
  0         ?  ?  ?  ?
  1         a1 a2 a3 a4
  2         b1 b2 b3 b4
```

The results show that FASTBLOCK behaves in the same manner as BLOCK however from the user point of view FASTBLOCK only requires two simulation time units. In the transformations such regions of hierarchical timing will be transformed to a common time frame and sample-and-hold constructs would be appropriately placed in the circuit. A user could of course have written the circuit with sample-and-hold constructs from the outset, however the use of the hierarchical timing means that such detail can be hidden.

5 Conclusions

In this document we have formally defined, through the use of predicate calculus, the semantics of all ELLA delays and the new retiming primitive. Examples of the new hierarchic timing concept have also been given.

6 Acknowledgements

The authors would like to acknowledge the support of the ESPRIT 'SPRITE' project 2260 and the IED Formal Verification project 4/1/1357. The authors would also like to acknowledge Praxis Electronic Design for their contribution to the retiming enhancement.

7 References

1. M. Hill, Private Communication
2. W. Smits, Private Communication
3. W. Smits, Private Communication
4. P. Rouse, "Requirements Specification for Timescaling", Praxis document ref. no. E.N0045.20.16 (also appeared as Sprite deliverable D3.3/Praxis/Y1-M6)
5. P. Rouse, "Specification for Timescaling", Praxis document ref. no. E.N0045.50.12
6. M. Hill, J. D. Morison, "Timescaling in ELLA", ESPRIT project 2260 deliverable D3.3/RSRE/Y1-M6, 1989
7. M. Hill, E. V. Whiting, "ELLA Timing", ESPRIT project 2260 deliverable D3.3/RSRE/Y1-M12, 1989
8. C. Sheers, "User Manual for the S2C Silage to C Compiler", Internal Report, IMEC 1988.
9. R. Stoll, "Set Theory and Logic", W. H. Freeman and Company, 1963.
10. R. L. Goodstein, "Mathematical Logic", Leicester University Press, 1965.

A Predicate Notation

We start by considering the following English sentence

Every rational number is a real number (1)

which may be translated as

For every x, if x is a rational number, then x is a real number (2)

In ordinary grammar, "is a real number" is known as the predicate of (1). In the translation (2) the added predicate "x is a rational number" replaces the common noun "rational number". Using "Q(x)" for "x is a rational number" and "R(x)" for "x is a real number", we may symbolise (2) as

$$\forall x,\ Q(x) \Longrightarrow R(x) \tag{3}$$

Hence the statement "3 is a rational number" may be symbolised by

Q(3)

Now the sentence

Some real numbers are rational

may be translated as

For some x, x is a real number and x is a rational number

and using the predicates introduced above, this may be symbolised as

$$\exists x,\ R(x) \bigwedge Q(x) \tag{4}$$

Similarly the predicate

$$\forall x,\ R(x) \Longrightarrow Q(x) \tag{5}$$

says that if x is a real number then it is also a rational number. This has the same meaning as

$$\exists x,\ \neg R(x) \bigvee Q(x) \tag{6}$$

which says

There is something which 'is not a real number' or 'is a rational number'

Now consider the predicate

$$(\forall x)\,[\,Q(x) \Longrightarrow R(x)\,] \wedge (\forall x)\,[\,Q(x) \Longrightarrow S(x)\,] \tag{7}$$

where S(x) stands for "is an irrational number". Then this predicate can be rewritten in the following form

$$(\forall x)\,[\,Q(x) \Longrightarrow R(x) \mid S(x)\,] \tag{8}$$

which stands for

$$(\forall x)\ IF\ Q(x)\ THEN\ R(x)\ ELSE\ S(x) \tag{9}$$

Note that although R(x) is true when Q(x) is false it is not deducible from this predicate.

THIS PAGE IS LEFT BLANK INTENTIONALLY

REPORT DOCUMENTATION PAGE

DRIC Reference Number (if known) .....

Overall security classification of sheet .....Unclassified.....

(As far as possible this sheet should contain only unclassified information. If it is necessary to enter classified information, the field concerned must be marked to indicate the classification eg (R), (C) or (S).

Originators Reference/Report No.	Month	Year
MEMO 4436	NOVEMBER	1990

Originators Name and Location
RSRE, St Andrews Road
Malvern, Worcs WR14 3PS

Monitoring Agency Name and Location

Title
FORMAL SEMANTIC DEFINITION OF ELLA TIMING

Report Security Classification	Title Classification (U, R, C or S)
UNCLASSIFIED	U

Foreign Language Title (in the case of translations)

Conference Details

Agency Reference	Contract Number and Period

Project Number	Other References

Authors	Pagination and Ref
HILL, M G; WHITING, E V; MORISON, J D	18

Abstract

A formal definition of the ELLA timing model is presented using the notation of predicate calculus. The definition of the new sample primitive used for the retiming of ELLA circuits is also given. Examples of retimed circuits are provided.

Abstract Classification (U,R,C or S)
U

Descriptors

Distribution Statement (Enter any limitations on the distribution of the document)
UNLIMITED

THIS PAGE IS LEFT BLANK INTENTIONALLY